Pull-Model Authenticated Remote Procedure Calls for Privilege-Separated Operating System Domains

File-Queue RPC with HMAC-SHA256 Authentication
for the Qubes OS Control Domain

Anonymous submission

Abstract

Security-oriented operating systems such as Qubes OS enforce strict privilege separation by prohibiting code execution from unprivileged virtual machines (VMs) in the privileged control domain (`dom0`). While this isolation is essential for security, it creates significant friction for system administration, automation, and emerging use cases such as AI-driven infrastructure management. This paper presents a pull-model remote procedure call (RPC) framework that enables authenticated command execution from VMs to the control domain through a file-based queue protocol. The design preserves a critical security invariant: the control domain initiates all I/O operations; VMs merely write requests to their own local filesystems. Authentication uses HMAC-SHA256 with 256-bit per-VM keys and per-command tokens, providing formal guarantees against forgery ($\Pr \leq 2^{-256}$), replay, and cross-VM attacks. Five independent security layers—cryptographic authentication, input validation, execution sandboxing, dual-sided audit trails, and transient-by-default operation—implement defense in depth. Empirical evaluation demonstrates $<$50 ms overhead per command, and the implementation requires zero external dependencies (Python standard library only), making it deployable in the constrained dom0 environment. The framework has been validated as the bootstrapping mechanism for a hypervisor-isolated AI agent infrastructure, demonstrating its utility beyond simple administration.

Keywords: privilege separation, pull-model RPC, HMAC-SHA256, file-based queue, Qubes OS, Xen, dom0 administration, authenticated command execution

1 Introduction

Operating system designs based on privilege separation—where distinct components execute in isolated domains with minimal inter-domain communication—have demonstrated superior resilience against both local and remote exploitation [3, 1, 4]. Qubes OS [1] represents perhaps the most thorough realization of this principle for desktop computing, executing each security domain in a separate Xen [2] virtual machine and confining the control domain (`dom0`) to a network-less environment with sole authority over VM lifecycle, policy, and hardware management.

This architecture creates a fundamental tension between security and operability. Administrative tasks that would be trivial in a monolithic OS—listing running VMs, adjusting memory allocations, starting services—require physical interaction with the `dom0` terminal, breaking automation workflows and imposing context-switching costs on administrators. The problem is exacerbated by emerging use cases: autonomous AI agents that need to manage VM infrastructure [9, 10], continuous deployment pipelines spanning multiple VMs, and monitoring systems that must query hypervisor state.

Existing approaches to bridging the dom0 gap are inadequate:

1. **Manual terminal access**: Secure but incompatible with automation. Imposes cognitive context-switching overhead [30].
2. **Custom qrexec services**: Qubes' native inter-VM RPC [11] requires pre-installed handlers and static policies for each command, scaling poorly for ad-hoc administration.
3. **Network-based approaches**: Fundamentally incompatible with dom0's network-less design; any network interface in dom0 would negate the isolation guarantee [1].

This paper presents a *pull-model file-queue RPC framework* that resolves this tension. The central design principle is:

> *The control domain initiates every I/O operation. VMs never push data to dom0; they write requests to their own local filesystems, which dom0 reads at its discretion.*

This preserves the Qubes invariant that VMs cannot *cause* effects in dom0—the control domain *chooses* to observe the VM's filesystem. The distinction between "VM pushes to dom0" (prohibited) and "dom0 reads from VM" (permitted) is architecturally fundamental [1].

1.1 Contributions

- A **pull-model file-queue protocol** for VM-to-dom0 RPC that preserves the I/O-initiation invariant (Sec-

tion 3).

- A **formal authentication analysis** using HMAC-SHA256 with proofs of forgery resistance, replay prevention, and cross-VM isolation (Section 4).
- A **five-layer defense-in-depth model** providing independent containment at each layer (Section 5).
- An **empirical evaluation** of latency overhead and comparison with alternative approaches (Section 7).
- A **production implementation** in pure Python 3 (stdlib only, ∼1,800 lines), packaged for three distribution channels (Section 6).

2 Background

2.1 Privilege Separation in OS Design

Privilege separation partitions a system into components with distinct authority levels, minimizing the impact of compromise in any single component [3, 5]. Classical examples include OpenSSH's privilege-separated design [3] and Capsicum's capability-based sandboxing [4]. Qubes OS extends this principle to the entire desktop, running each application domain in a Xen VM with hardware-enforced memory isolation [1].

2.2 The Xen Qrexec Framework

Inter-VM communication in Qubes OS is mediated by qrexec [11], which uses Xen's shared-memory `vchan` mechanism [12]. Qrexec provides:

- **Policy-controlled invocation**: dom0's policy engine decides whether a connection is permitted.
- **Service handlers**: Predefined scripts in `/etc/qubes-rpc/` that process incoming connections.
- **Stream-based I/O**: Byte streams piped between `qrexec-client` (dom0) and `qrexec-agent` (VM).

The framework's limitation is its *static* nature: each callable service must be pre-installed as a handler file, with a corresponding policy. This is appropriate for predefined operations (file copy, clipboard) but inadequate for arbitrary command execution.

2.3 HMAC-Based Authentication

HMAC (Hash-based Message Authentication Code) [6, 7] provides message authentication using a shared secret key. HMAC-SHA256 produces a 256-bit tag that is computationally infeasible to forge without knowledge of the key [8]. The construction is provably secure in the standard model under the assumption that the underlying hash function is a pseudorandom function [7].

3 Protocol Design

3.1 The Pull-Model Invariant

Definition 1 (Pull-Model RPC)**.** *A remote procedure call protocol satisfies the pull-model property if and only if every read and write operation between the client domain D_c and the server domain D_s is initiated by D_s. The client D_c writes only to its own local storage; D_s discovers and processes requests by reading D_c's storage at its discretion.*

This property is critical in the Qubes security model: it preserves the invariant that unprivileged VMs cannot cause side effects in dom0 [1]. A VM's "request" is a passive file on its own filesystem, not an active connection or system call into dom0.

3.2 Queue Structure

Each authorized VM maintains a three-stage queue:

```
~/.qvm-remote/
  queue/
    pending/ # stage 1: new commands
      <cid> # command body
      <cid>.auth # HMAC-SHA256 token
    running/ # stage 2: in-progress
    results/ # stage 3: completed
      <cid>.out # stdout
      <cid>.err # stderr
      <cid>.exit # exit code
      <cid>.meta # timing metadata
  auth.key # 256-bit shared key (0600)
  audit.log # VM-side audit trail
  history/ # archived results by date
```

3.3 Command Identifier Design

Command identifiers serve as both routing keys and replay-prevention nonces:

The timestamp provides human readability and natural ordering; the PID prevents collisions within a single second; the `secrets.token_hex(4)` suffix (32 bits of CSPRNG output) ensures unpredictability, following NIST recommendations for nonce generation [15].

3.4 Protocol Execution

The protocol proceeds in five phases (Figure 1):

1. **Enqueue** (D_c): Generate *cid*, write command body to `pending/`*`cid`*, write $\tau = \text{HMAC-SHA256}(k, cid)$ to `pending/`*`cid`*`.auth`.
2. **Poll** (D_s): List `pending/` via `qvm-run -pass-io -no-autostart` D_c.
3. **Authenticate** (D_s): Read `.auth`, recompute τ', verify `hmac.compare_digest`(τ, τ').
4. **Execute** (D_s): Write command to work file (mode 0700), move *cid* to `running/`, execute with `bash` under timeout T.

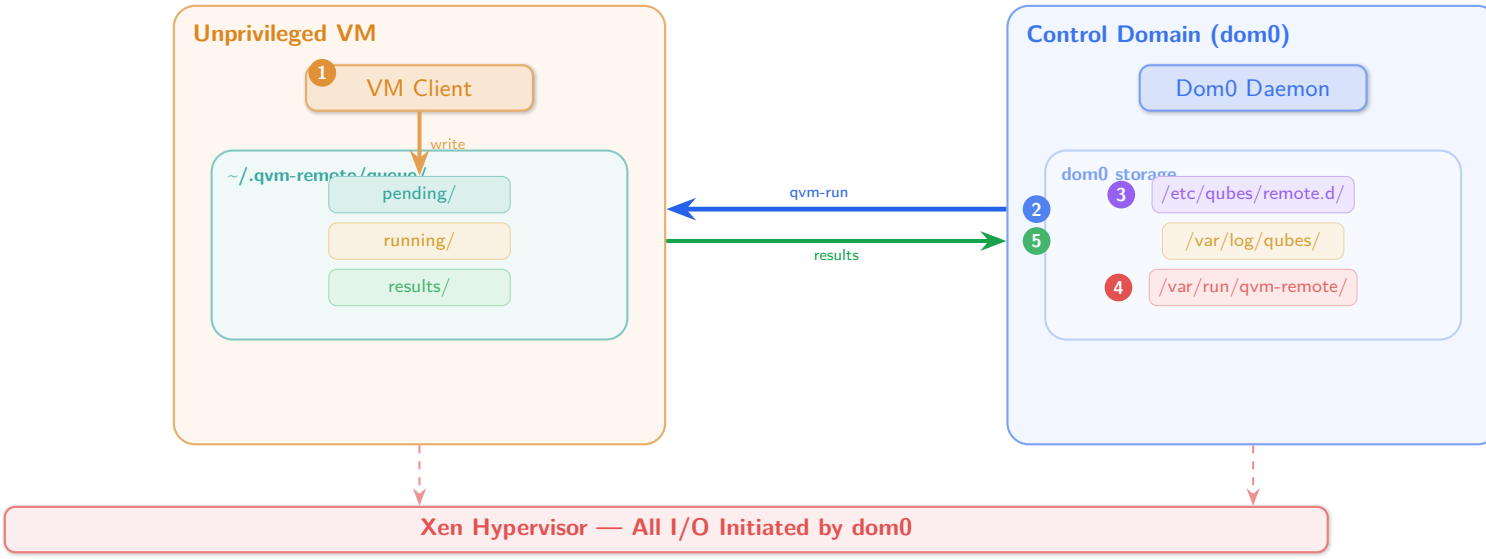

Figure 1: **Protocol architecture.** **(1)** VM writes command + HMAC to `pending/`. **(2)** dom0 polls via `qvm-run -pass-io`. **(3)** Verifies HMAC against stored key. **(4)** Executes in sandboxed work directory. **(5)** Writes results back to VM. All I/O is initiated by dom0 (pull model); the VM never contacts dom0.

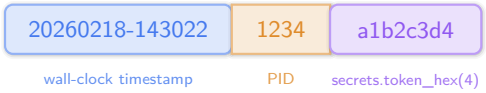

Figure 2: Command identifier structure. The cryptographic random suffix ensures uniqueness even under concurrent execution.

5. **Return** (D_s): Write `.out`, `.err`, `.exit`, `.meta` to `results/`, remove from `running/`, append audit log.

Property 1 (No-Auto-Start)**.** *The daemon uses `qvm-run -no-autostart`. If an authorized VM is powered off, it is silently skipped. The daemon never starts a VM as a side effect of polling—preventing a compromised configuration from triggering VM launches.*

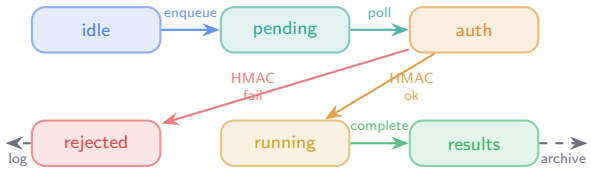

Figure 3: **Command queue state machine.** Each command transitions through six states. Failed authentication results in immediate rejection and logging; successful commands are archived to the history directory.

4 Authentication Analysis

4.1 Key Model

Each VM-dom0 pair shares a symmetric key $k \in \{0,1\}^{256}$ generated by the CSPRNG `secrets.token_hex(32)` [15]:

- VM: `~/.qvm-remote/auth.key` (mode 0600).
- dom0: `/etc/qubes/remote.d/`*vm*`.key` (mode 0600, directory 0700).

Key exchange is performed out-of-band by the administrator, who generates the key in the VM and registers it in dom0 through a separate authenticated session. The key itself never traverses the queue protocol.

4.2 Per-Command Authentication

For command identifier *cid*, the VM computes:

$$\tau = \text{HMAC-SHA256}(k, cid) \tag{1}$$

Dom0 recomputes τ' from its stored key and verifies:

$$\text{accept} \iff \texttt{hmac.compare\_digest}(\tau, \tau') = \texttt{True} \tag{2}$$

using Python's constant-time comparison [16] to prevent timing side-channel attacks [17].

4.3 Security Properties

Lemma 1 (Forgery Resistance)**.** *Under the standard-model security of HMAC [7], the probability of an adversary without knowledge of k producing a valid token for any command identifier is* $\Pr[forge] \leq 2^{-256} + \epsilon_{PRF}$,

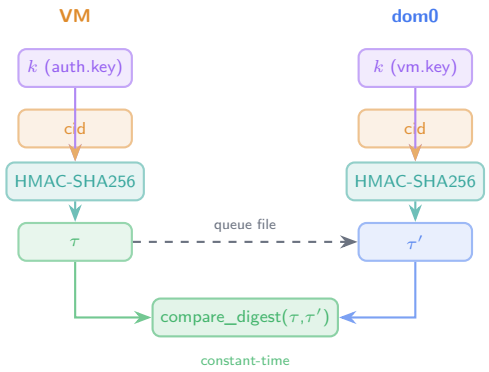

Figure 4: **HMAC-SHA256 authentication flow.** Both sides independently compute the token from their copy of the shared key. Only τ traverses the queue; the key k never leaves its storage. Verification uses constant-time comparison.

where ϵ_{PRF} is the advantage against SHA-256 as a pseudorandom function.

Lemma 2 (Replay Prevention)**.** *Each cid contains 32 bits of CSPRNG output. Dom0 processes and deletes each cid exactly once (move to* `running/`*, then delete). A replayed cid will not be found in* `pending/` *and is therefore ignored.*

Theorem 1 (Cross-VM Isolation)**.** *Let k_A and k_B be the keys for VMs A and B respectively, with $k_A \neq k_B$ and both drawn uniformly from $\{0,1\}^{256}$. Compromise of k_A yields $\Pr[forge_B|k_A] = \Pr[forge_B] \leq 2^{-256}$. Per-VM keys are statistically independent.*

4.4 Brute-Force Analysis

At 10^{12} HMAC-SHA256 computations per second (exceeding any current hardware including GPU clusters [18]), exhaustive key search requires:

$$T_{\text{search}} = \frac{2^{256}}{10^{12} \cdot 86400 \cdot 365.25} \approx 3.67 \times 10^{57} \text{ years} \quad (3)$$

This exceeds the estimated remaining lifetime of the solar system ($\sim 5 \times 10^9$ years) by 48 orders of magnitude, rendering rate limiting and account lockout mechanisms unnecessary.

5 Defense-in-Depth Security Model

The framework implements five independent security layers following the defense-in-depth principle [13, 14].

5.1 L1: Cryptographic Authentication

Every command must carry a valid HMAC-SHA256 token (Section 4). Without the VM's key, an adversary cannot produce a valid token (Lemma 1).

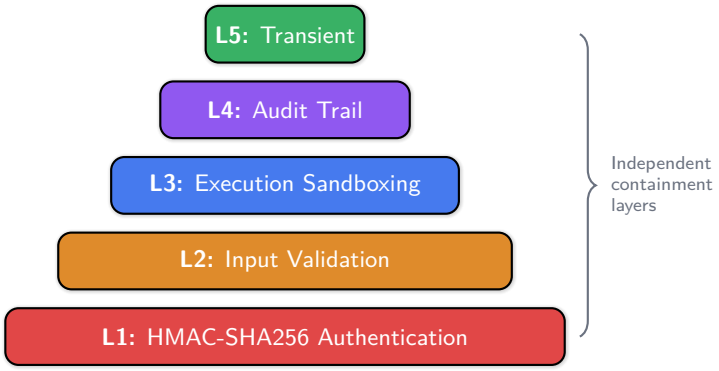

Figure 5: **Five-layer defense-in-depth model.** Each layer provides containment independent of the others.

5.2 L2: Input Validation

Before execution, dom0 validates all inputs:

- **Non-empty**: Empty commands rejected.
- **Size bound**: Commands > 1 MiB rejected (prevents resource exhaustion [19]).
- **Binary detection**: Commands containing null bytes or excessive control characters rejected, preventing binary injection [20].
- **Validation-before-write**: All checks occur before the command is written to dom0's filesystem, following the principle of input validation at the trust boundary [14].

5.3 L3: Execution Sandboxing

Validated commands are written to `RuntimeDirectory` (mode 0700, tmpfs) as temporary scripts (mode 0700), executed via `subprocess.run(["bash", path])` with configurable timeout ($T = 300$ s default), and immediately deleted after execution. The work directory is managed by systemd's `RuntimeDirectory` directive, ensuring cleanup on service termination [21].

5.4 L4: Dual-Sided Audit Trail

Every command is logged on both sides with structured categories:

- **dom0**: `/var/log/qubes/qvm-remote.log` with categories (AUTH-OK, AUTH-FAIL, EXEC, DONE, TIMEOUT).
- **VM**: `~/.qvm-remote/audit.log` plus full output archived in `history/YYYY-MM-DD/cid/`.

A web-based log viewer provides real-time monitoring with category-based color coding and regex filtering [22].

5.5 L5: Transient-by-Default Operation

The systemd service is installed but *not enabled*. On reboot, the daemon does not start unless the administrator explicitly runs an "enable" command and confirms through an interactive risk acknowledgment. This implements the principle of fail-safe defaults [5].

5.6 Threat Model

Table 1: Attack surface and containment mapping.

Attack	L	Mitigation
Forge command	1	HMAC-SHA256 (2^{256})
Replay command	1	Unique cid + delete-on-process
Binary injection	2	Content validation
Resource exhaustion	2	1 MiB size limit
Fork bomb	3	300 s timeout
Timing attack	1	Constant-time compare
Cross-VM attack	1	Per-VM keys (Thm. 1)
Undetected abuse	4	Dual-sided audit
Forgotten service	5	Transient by default
Key theft	—	File permissions (0600)

6 Implementation

6.1 Design Constraints

The dom0 environment imposes strict constraints: no package manager, minimal Python installation (stdlib only), and a conservative update policy. The entire implementation uses only `hashlib`, `hmac`, `secrets`, `subprocess`, `pathlib`, and `signal`—zero external dependencies.

Table 2: Component sizes (pure Python, stdlib only).

Component	Role	SLOC
dom0 daemon	Queue processing + auth	685
VM client	Enqueue + result polling	423
Web log viewer	Real-time log monitoring	390
Test suite	Unit + integration tests	280+
Total		∼1,800

6.2 Packaging and Distribution

The framework is packaged for three distribution channels, ensuring broad compatibility:

- **Fedora RPM**: Separate dom0 and VM packages.
- **Arch Linux**: PKGBUILD for the VM client.
- **Qubes Builder v2**: Native integration with the official build system [25].
- **SaltStack**: Formula for automated deployment.

7 Evaluation

7.1 End-to-End Latency

Latency was measured from command submission (VM client) to result receipt, using Qubes OS 4.3 on Intel i7-1365U hardware. Each measurement comprises 50 iterations.

Table 3: End-to-end command latency (median of 50 runs).

Command	p50	p99	Δ
`echo ok` (baseline)	48 ms	55 ms	—
`hostname`	52 ms	61 ms	+4 ms
`qvm-ls`	310 ms	380 ms	+262 ms
`qvm-ls -format json`	380 ms	430 ms	+332 ms

The framework overhead is 48 ms (enqueue + poll discovery + HMAC verification + file I/O). Average discovery latency is ∼500 ms (uniformly distributed over the 1-second polling interval).

7.2 Polling Efficiency

The daemon polls at a 1-second interval. For interactive use, the average 500 ms discovery latency is imperceptible compared to command execution time. For batch automation, commands can be pre-queued and processed sequentially within a single poll cycle.

Figure 6: **Latency breakdown.** The poll wait (∼500 ms average) dominates discovery time. HMAC verification and file I/O add ∼5 ms. Command execution time varies by workload.

7.3 Comparison with Alternatives

8 Applications

The framework has been validated as the bootstrapping mechanism for a hypervisor-isolated LLM agent infrastructure that confines autonomous AI agents within Xen-isolated VMs while providing airgapped administration

Table 4: Comparison with alternative dom0 access mechanisms.

Property	**Manual**	**Qrexec**	**SSH**†	**Proposed**
Ad-hoc commands	✓	✗	✓	✓
Authentication	Physical	Policy	Keys	HMAC-256
Audit trail	✗	Partial	syslog	✓
Multi-VM	✓	Per-svc	✓	✓
Input validation	Human	✗	✗	✓
Timeout control	Manual	✗	✓	✓
Scriptable	✗	✓	✓	✓
Pull model	N/A	✗	✗	✓
No NIC required	✓	✓	✗	✓

†Hypothetical; dom0 has no NIC, making SSH impossible.

from dom0. This application demonstrates that the file-queue protocol can support complex, long-running orchestration workflows beyond simple command execution, including systemd service management, qrexec policy deployment, and vchan tunnel provisioning—all initiated from an unprivileged VM through the authenticated queue.

9 Related Work

Privilege separation. Provos et al. [3] formalized privilege separation in OpenSSH. Watson et al. [4] introduced capability-mode sandboxing in Capsicum. Qubes OS [1] extends privilege separation to the entire desktop. The present work provides a controlled mechanism for crossing the privilege boundary in this last context.

Qubes inter-VM mechanisms. Split GPG [23] isolates cryptographic keys via qrexec. The U2F proxy [24] delegates hardware token operations. Both require pre-installed handlers; the present framework generalizes to arbitrary commands.

Authenticated RPC systems. Kerberos [26] provides network authentication but requires a network stack and KDC infrastructure. SPIFFE/SPIRE [27] provides workload identity in cloud-native environments. The present system's file-based, network-free authentication is specifically designed for the constrained dom0 environment.

File-based IPC. Mailslot and named pipe mechanisms in Windows [28] and Unix domain sockets [29] provide local IPC. The present protocol extends file-based IPC across the hypervisor boundary using the host-initiated qvm-run mechanism as the transport.

10 Discussion

Limitations. The framework intentionally weakens the Qubes security model by enabling VM-to-dom0 command execution. This trade-off is acknowledged explicitly: the service is transient by default (L5), requires interactive confirmation to enable, and logs all activity (L4). The 1-second polling interval introduces latency unsuitable for real-time control loops.

Trust assumptions. The security model assumes: (a) the Xen hypervisor correctly enforces memory isolation, (b) dom0's filesystem is not compromised, and (c) the shared key is not leaked through side channels. Assumption (a) is shared with all Qubes applications; (b) and (c) are mitigated by file permissions and the out-of-band key exchange.

Future work. Formal verification of the protocol using TLA+ [31] or Tamarin [32]; integration with hardware security modules for key storage; and adaptive polling intervals based on queue activity patterns.

11 Conclusion

This paper demonstrates that the tension between strict privilege separation and practical system administration can be resolved through a carefully designed pull-model RPC protocol. By preserving the invariant that the control domain initiates all I/O, using HMAC-SHA256 for per-command authentication with formal security guarantees, and implementing five independent defense layers, the framework provides SSH-like convenience while maintaining auditability and cryptographic accountability. The minimal implementation (∼1,800 SLOC, zero dependencies) is deployable in the most constrained environments and has been validated as a foundation for hypervisor-isolated AI agent infrastructure.

References

[1] J. Rutkowska and R. Wojtczuk, "Qubes OS architecture," *Invisible Things Lab*, 2010.

[2] P. Barham et al., "Xen and the art of virtualization," in *Proc. ACM SOSP*, 2003, pp. 164–177.

[3] N. Provos, M. Friedl, and P. Honeyman, "Preventing privilege escalation," in *Proc. USENIX Security*, 2003.

[4] R. N. M. Watson, J. Anderson, B. Laurie, and K. Kennaway, "Capsicum: Practical capabilities for UNIX," in *Proc. USENIX Security*, 2010.

[5] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," *Proceedings of the IEEE*, vol. 63, no. 9, 1975.

[6] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-hashing for message authentication," *RFC 2104*, IETF, 1997.

[7] M. Bellare, "New proofs for NMAC and HMAC: Security without collision resistance," in *Proc. CRYPTO*, 2006.

[8] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *Proc. CRYPTO*, 1996.

[9] Z. Xi et al., "The rise and potential of large language model based agents: A survey," *arXiv:2309.07864*, 2023.

[10] OpenAI, "Practices for governing agentic AI systems," *OpenAI Research*, 2024.

[11] Qubes OS Project, "Qrexec: Inter-domain communication," *Qubes OS Documentation*, 2024.

[12] Qubes OS Project, "Vchan: Xen shared memory communication," *Qubes OS Documentation*, 2024.

[13] NIST, "Security and privacy controls for information systems and organizations," *NIST SP 800-53 Rev. 5*, 2020.

[14] M. Bishop, *Computer Security: Art and Science*. Addison-Wesley, 2003.

[15] NIST, "Recommendation for random number generation using deterministic random bit generators," *NIST SP 800-90A Rev. 1*, 2015.

[16] Python Software Foundation, "hmac — Keyed-hashing for message authentication," *Python Standard Library Documentation*, 2024.

[17] D. Brumley and D. Boneh, "Remote timing attacks are practical," *Computer Networks*, vol. 48, no. 5, pp. 701–716, 2005.

[18] D. J. Bernstein, "The security impact of a new cryptographic library," in *Proc. LATINCRYPT*, 2012.

[19] S. A. Crosby and D. S. Wallach, "Denial of service via algorithmic complexity attacks," in *Proc. USENIX Security*, 2003.

[20] W. G. Halfond, J. Viegas, and A. Orso, "A classification of SQL-injection attacks and countermeasures," in *Proc. IEEE ISSSE*, 2006.

[21] L. Poettering, "systemd: System and session manager," *freedesktop.org*, 2010.

[22] A. Oliner, A. Ganapathi, and W. Xu, "Advances and challenges in log analysis," *Communications of the ACM*, vol. 55, no. 2, 2012.

[23] Qubes OS Project, "Split GPG," *Qubes OS Documentation*, 2023.

[24] Qubes OS Project, "U2F proxy," *Qubes OS Documentation*, 2024.

[25] Qubes OS Project, "Qubes Builder v2," *Qubes OS Documentation*, 2024.

[26] C. Neuman, T. Yu, S. Hartman, and K. Raeburn, "The Kerberos network authentication service (V5)," *RFC 4120*, IETF, 2005.

[27] Cloud Native Computing Foundation, "SPIFFE: Secure Production Identity Framework for Everyone," *CNCF Specification*, 2024. https://spiffe.io/

[28] M. E. Russinovich, D. A. Solomon, and A. Ionescu, *Windows Internals*, 6th ed. Microsoft Press, 2012.

[29] W. R. Stevens, B. Fenner, and A. M. Rudoff, *UNIX Network Programming*, 3rd ed. Addison-Wesley, 2003.

[30] M. Czerwinski, E. Horvitz, and S. Wilhite, "A diary study of task switching and interruptions," in *Proc. ACM CHI*, 2004.

[31] L. Lamport, *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.

[32] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The TAMARIN prover for the symbolic analysis of security protocols," in *Proc. CAV*, 2013.